Relational Databases and SQL

Research Thesis

Presented in partial fulfillment of the requirements for graduation *with research distinction* in the undergraduate colleges of The Ohio State University.

by

Jiajie Shen

The Ohio State University
April 2020

Project Advisor: Professor Rajiv Ramnath, Department of Computer Science and Engineering

Table of Contents

Acknowledgement

First of all, I would like to thank Dr. Rajiv Ramnath for giving me this fantastic opportunity to get involved with a research project. Through this project, I have improved my studying ability a lot. Also, I want to thank my previous academic advisor Dr. Rodica Barbu who supported me throughout my academic career and encouraged me to be an excellent student. I would also like to thank my parents and friends who strongly support me through my 4 year undergraduate education. Finally, I want to thank the Buckeye nation which is very supportive and gives me so many resources to connect to the world.

Chapter 1 - An Introduction to Relational Databases

The first chapter talks about some basic and interesting facts about databases. It basically gives some general definitions and makes everything clear in the first place. The first important point is that databases are not file sets. Files system is for storing most likely unrelated data. In contrast, a database is used for storing structured and related data. The following two tables is an example of unrelated data [1].

Phone_id	Phone_Name	Alpha
67382	Apple	A
46732	Google	G
57346	Samsung	S

Customer_id	First_Name	Last_Name
3462764343254	Jack	Utopia
4895673457345	Mike	Yoseline
8297539487549	Mark	Scott

The following is an example of relational data: SSN is the primary key for both of the tables.

Age	**Name**	**SSN**	**Employer**
27	Mark	392-89-89238	OSU
45	Andrew	489-38-78945	JP Morgan

Customer	Purchased products	Products_ID	SSN
Mark	milk	37289472893472	392-89-89238
Hannah	avocado	81271894798322	382-78-78493

The Data Declaration Language (DDL) in SQL is used to deal with how data should exist at a logical level in a database. Data Declaration Language contains commands like create(is used to create the database), drop(delete objects in the database), alter(alter structure in the database), comment(add comments), rename(is used to rename an object).

Also, There are two easy mistakes that many beginners will make: They think table is a file and table is a spreadsheet. Here is why. The most important difference is that databases have data integrity which means we cannot store

different types of data in the same database. But in the spreadsheets, we can create many different types of data as we wish. These are the very common mistakes that we will make while studying the SQL programming language. We should try to avoid making these mistakes so that we can better understand how SQL works.

A field within a record is defined by the program which reads it. A column in a row in a table is defined by the database schema. The data types in a column are scalar all the time. We can use the check() function to test whether a table is empty or not. If the table is empty then check() will turn out true; if the table is not empty then check() will turn out to be false.

Chapter 2 - Tables

This chapter is entitled “transactions and concurrency control”. We will talk about something related to this. First, the word A Atomicity: This means that the entire transaction becomes consistent in the SQL database or nothing in the transaction becomes consistent. For example, a specific car can only be filled up with a certain kind of gas. Like the Honda pilot only needs #87 gas. We can simply think this as true/false statements. The result can be either true or false, it can be only in 2 outcomes. Similarly, the transaction of databases can be either consistent or inconsistent. The word “consistent” in SQL means that all of the data integrity constraints, relational integrity constraints, and all other constraints are true. For instance, the editor inserted one thousand rows into the table and one of the rows has a little referential problem. Then the whole database will do an automatic error check.

Concurrency control is just like it means in word: there is a situation where so many people are accessing the same database at the same time. How can we manage that in order to let those people without running into each other? The book has given us several explanations.

Here is an example of concurrency control:

A	B	C
34	F	10
2	E	8

Table 1

Person1: UPDATE Table1 Set A=3 Where A=2

Person2: UPDATE Table1 Set A=5 Where A=2

Both persons want to update the table but only the first person can do that.

The isolation levels can make sure that each transaction will be executed or not. In this way, the new changes in the database will not be lost. If the database machine detects some errors or is unable to complete multiple concurrent transactions, it will just generate the ROLLBACK which is to check whether there are errors in the system. There are 3 specific isolation levels: Serializable isolation level is 100% to produce the same results as the concurrent transactions have had if they were done in some serial order(100% of the system will be shut down and check the error, it is the most strict one); Repeatable read isolation level is 100% to maintain the same image of the database(It is the most popular method); Read committed isolation level can let transactions see rows which other transactions

commit when this session is still running(this method is let the users still see rows instead of shutting down the whole system).

Pessimistic concurrency control assumes that transactions conflict with each other. So it is using something called “lock” in the database, for better understanding of this, it is just like the toilet showing “occupied”. It is not used very often because if we lock down the whole table then all the users will need to wait for it. The pessimistic concurrency is usually used for maintenance. Page locking is the compromise between the whole table lock down and row locking. But the performances of these also depend on the data distribution in the specific situations.

Chapter 3 - Keys

This chapter is mainly talking about tables, tables consisting of multiple rows or zero row. If it is a zero row table then it is meaningless which means we do not have data at all. Tables are the most essential things in SQL, and it is the only one structure in SQL.

There are three important things we need to know about tables: the tables have no specific order and they are referenced by table names(the tables in the SQL is not like the tables in excel, tables in the SQL don't have a particular order); the rows have no specific order, they are referenced by a thing called key(Again, rows of tables in excel have specific order but rows of tables in SQL don't have a specific order. For example, a salary table contains rows of employee name, amount of salary, age, gender and so on, but all the rows are not in order); The columns have no specific order in the row, they are referenced by column names, Each column has two things which are: a data type and a name. (back in the salary table, it contains columns of employee name, amount of salary, age, gender and so on. All the columns don't have a specific order but users can always find them by searching the data type and the name of it.)

The following code teaches us how to create a table in the SQL database. This is the most basic format and we can always use other different formats.

```
► CREATE TABLE PRODUCT (
►     ProductID       INTEGER    PRIMARY KEY,
►     Name            CHAR (25),
►     Description       CHAR (30),
►     Category       CHAR (15),
►     VendorID       INTEGER,
►     VendorName       CHAR (30) ) ;
```

The temporary tables are used to store the intermediate results. For example, you will need 5 steps to calculate the final results from a dataset. Then you can store the results from step 1-4 in the temporary table and then use those results to calculate the final answer in step 5. This is a very useful and helpful method when we are dealing with the real-world problem.

Column is one of the most important parts in the SQL query. Each column has to have a data type. And we have 3 major data types: character, numeric and temporal. Here is a simple example, < Employee ID > < numeric>. This example shows the column's name is Employee ID and all the data associated with it is in numeric data type.

The sequence in SQL is an ordered list of whole numbers. Sometimes, this term is kind of easily messed up with the term series. Series is totally different from sequence, series is used more in the mathematical area. Let me give an easy

example to illustrate the sequence in SQL. We have a column called <Employee age> and all the specific ages in the sequence will be ordered. For example, 23,25,27,29,31,33,35 and so on. All the ages of the employees will be ordered from the smallest to the largest.

Here is an example of sequence in SQL: it is a sequence from 1000 to 0 and it is like 1000, 990, 980, 970, 960…….20, 10, 0.

- CREATE SEQUENCE 1000-0
- start with 1000
- increment by -10
- minvalue 0
- maxvalue 1000
- cycle;

Chapter 4 - Normalization

We are introducing keys in this chapter. The first type of key is called natural key, it is a subset that has a unique identifier in the table. Like the QR code for specific products, labels for your FedEx package and so on. The second type of key is called artificial key, it is an extra feature in the table which can be seen by the users. For example, an automobile selling store not only sells the cars with vin but also sells some repairing tools without labels. It is a wonderful tool for database designers to verify their codes by using this trusted source.

If we want to have a sorting network in the SQL database then we need Update statements. Here is an example to explain how that works.

```
BEGIN ATOMIC
-- Swap(a, b);
UPDATE Foobar
SET a = b, b = a
WHERE a > b;
-- Swap(d, e);
UPDATE Foobar
SET d = e, e = d
WHERE d > e;
-- Swap(c, e);
UPDATE Foobar
SET c = e, e = c
WHERE c > e;
-- Swap(c, d);
UPDATE Foobar
SET c = d, d = c
WHERE c > d;
-- Swap(a, d);
UPDATE Foobar
SET a = d, d = a
WHERE a > d;
-- Swap(a, c);
UPDATE Foobar
```

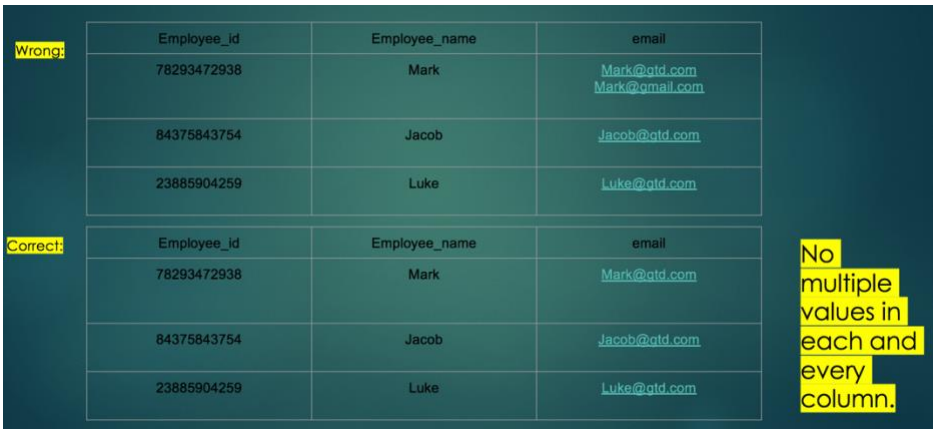
```
SET a = c, c = a
WHERE a > c;
-- Swap(b, e);
UPDATE Foobar
SET b = e, e = b
WHERE b > e;
-- Swap(b, d);
UPDATE Foobar
SET b = d, d = b
WHERE b > d;
-- Swap(b, c);
UPDATE Foobar
SET b = c, c = b
WHERE b > c;
END;
```

Where a, b, c, d, e are different items. It is like writing a simple java program, a, b, c, d, e supposed to be in an order from the smallest to the largest, but once there is an order change then we do need to swap the order in order to get the new sequence. The method is like the IF function in java programming language, if the condition is satisfied then the code inside of the branch will run. After running the whole program, all the items will be placed in the correct position.

Chapter 5 – Concurrency control and Transactions

We are going to talk about Normalization in this chapter. A normal form is to classify a SQL table depending on the functional dependencies. The FD has a simple definition: If I know an attribute's value then I should be able to determine the other attribute's value. For example, there is a table for the prices for different kinds of automobiles. If we know the vin number for the car then we have the access to see how much the car is.

Here is an example for First Normal Form(1NF):



We also have a convenient way to create tables with strings input. Here is an example:

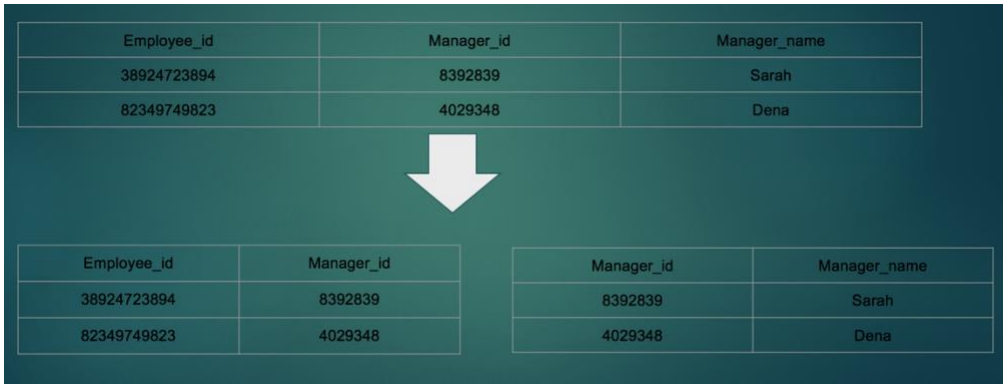
```
CREATE TABLE InStrings
(key_col CHAR(20) NOT NULL PRIMARY KEY,
```

```
input_string VARCHAR(275) NOT NULL);
INSERT INTO InStrings VALUES ('first', '22,36,867,898');
INSERT INTO InStrings VALUES ('second', '332,334,597,708');
```

In this example, we create 2 rows in the table. The first row starts with the word first and then comes with a number `22,36,867,898.` The second row starts with the word second and then comes with a number `332,334,597,708.`

Let's move on to the second normal form which is also known as 2NF. It is the form in the 1NF and no partial key dependencies. Let me give an example to illustrate the definition of the second normal form: we have two columns called department_name and professors_name, department_name is the key. We also have a column called advisor_name which is a subset of department_name. In this case, we cannot say that advisor_name depends on professors_name.

The following picture shows an example for the second normal form:



We also have the third normal form. There is one fact that does not exist in the 3NF: there are 3 columns in the table which are j, f, k. We have that j depends on f; f depends on k; so we know that j depends on k by a simple mathematical fact.

Here is an example for the 3NF: the original table needs to be split into two new tables in order to become the third normal form.

Employee_id	Employee_name	Manager_id	Manager_name	Employee_age
78293472938	Mark	8392839	Sarah	47
84375843754	Jacob	4029348	Dena	39

Employee_id	Employee_name	Employee_age
78293472938	Mark	47
84375843754	Jacob	39

Manager_id	Manager_name
8392839	Sarah
4029348	Dena

Bibliography

1. https://stackoverflow.com/questions/40660704/how-can-i-display-data-from-two-unrelated-tables/40660754

2. https://searchdatamanagement.techtarget.com/definition/relational-database